Práctica 2
Arboles Binarios y Heap

Ejercicio 1

En base a la implementación de árboles binarios proporcionada por la cátedra, complete la misma agregando los siguientes métodos:

- agregar(elemento) # Agrega elemento al árbol binario de búsqueda
- incluye(elemento) # Retorna True si elemento está en el árbol, False en caso contrario
- inorden() # imprime un recorrido inorden del árbol binario de búsqueda
- preorden() # imprime un recorrido preorden del árbol binario de búsqueda
- postorden() # imprime un recorrido postorden del árbol binario de búsqueda

Pruebe la implementación de árbol binario, armando un árbol para luego imprimirlos con los distintos recorridos (inorden, postorden y preorden).

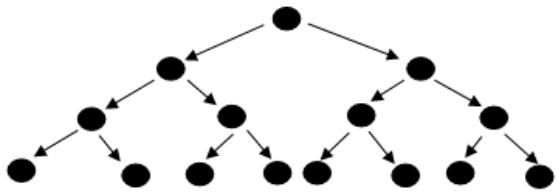
Ejercicio 2

Agregue a la clase ArbolBinario los siguientes métodos:

- contarHojas() #devuelve la cantidad de subárboles hojas del árbol receptor.
- entreNiveles(int n, m) # imprime el recorrido por niveles de los elementos del árbol receptor entre los niveles n y m (ambos inclusive). ($0 \leq n < m \leq$ altura del árbol)

Ejercicio 3

Una red binaria es una red que posee una topología de árbol binario lleno. Por ejemplo:



Los nodos que conforman una red binaria llena tiene la particularidad de que todos ellos conocen cuál es su retardo de reenvío. El retardo de reenvío se define como el período comprendido entre que un nodo recibe un mensaje y lo reenvía a sus dos hijos.
Su tarea es calcular el mayor retardo posible, en el camino que realiza un mensaje desde la raíz hasta llegar a las hojas en una red binaria llena. Cree una clase llamada RedBinariaLlena donde implementará lo solicitado en el método retardoReenvio():int.

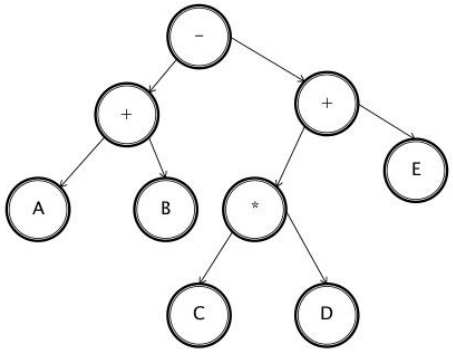
Ejercicio 4

Implemente una clase llamada ProfundidadDeArbolBinario que tiene como variable de instancia un árbol binario de números enteros y un método de instancia sumaElementosProfundidad(int p):int el cuál

devuelve la suma de todos los nodos del árbol que se encuentren a la profundidad pasada como argumento.

Ejercicio 5

Dado el siguiente árbol de expresión:



Indique las expresiones aritméticas resultantes de realizar una impresión en pre-orden, in-orden y post-orden.

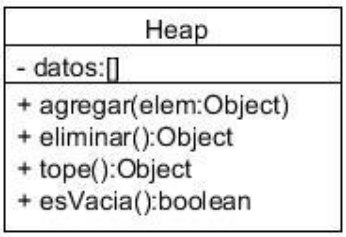
Ejercicio 6

Muestre las transformaciones que sufre una Max HEAP (inicialmente vacía) al realizar las siguientes operaciones:

a. Insertar 50, 52, 41, 54, 46
b. Eliminar tres elementos
c. Insertar 45, 48, 55, 43
d. Eliminar tres elementos.

Ejercicio 7

Considere la siguiente especificación de la clase **Heap**:



El método esVacia():bool devuelve **true** si no hay elementos para extraer.

El método agregar(elem) agrega un elemento en la heap y devuelve **true** si la inserción fue exitosa y **false** en caso contrario.

El método eliminar() elimina el tope y lo retorna.

El método tope() retorna el tope sin eliminarlo.

Además implemente un constructor en la clase **Heap** que tome dos parámetros, el primero un arreglo de Objects con el cual se crea e inicializa la Heap a partir de reordenar los elementos del mismo y el segundo un boolean que indique si se trata de una Max HEAP o Min HEAP (utilice **true** para indicar que es Max HEAP y **false** para Min HEAP).

Nota: Recordar tener en cuenta que la estructura a utilizar internamente para almacenar los elementos, tiene que tener un mecanismo de comparación entre objetos para poder evaluar las claves.

Ejercicio 8

Se desea implementar una clase **Impresora** que ofrezca los siguientes métodos:

- nuevoDocumento(documento: String): Encola un nuevo documento para imprimir.

- imprime(): Imprime por pantalla el documento almacenado más corto y lo elimina de memoria. Si no hay documentos en espera no hace nada.